UNIVERSITY OF EXETER

FACULTY OF ENVIRONMENT, SCIENCE AND ECONOMY

COMPUTER SCIENCE

ECMM461
High Performance Computing

Continuous Assessment 2

Date Set: 9 March 2023
Date Due: 29 March 2023
Return Date: 3 May 2023

This CA comprises 40% of the overall module assessment.
This is an individual exercise and your attention is drawn to the College and University guidelines on collaboration and plagiarism, which are available from the College website.
This is the coursework for this module and tests your understanding of how to find numerical solutions to partial differential equations, and parallel programming.

This continuous assessment comprises 40% of the overall module assessment.

1 The Manager-worker version of the Mandelbrot calculation

The Mandelbrot set is the set of complex numbers which contains the points for which the iteration

$$z_{i+1} = z_i^2 + C \tag{1}$$

remains finite, where C is a complex number and the initial value $z_0 = C$. This apparently simple relation leads a fractal structure (see figure 1 which shows the Mandlebrot set plotted on the complex plane).

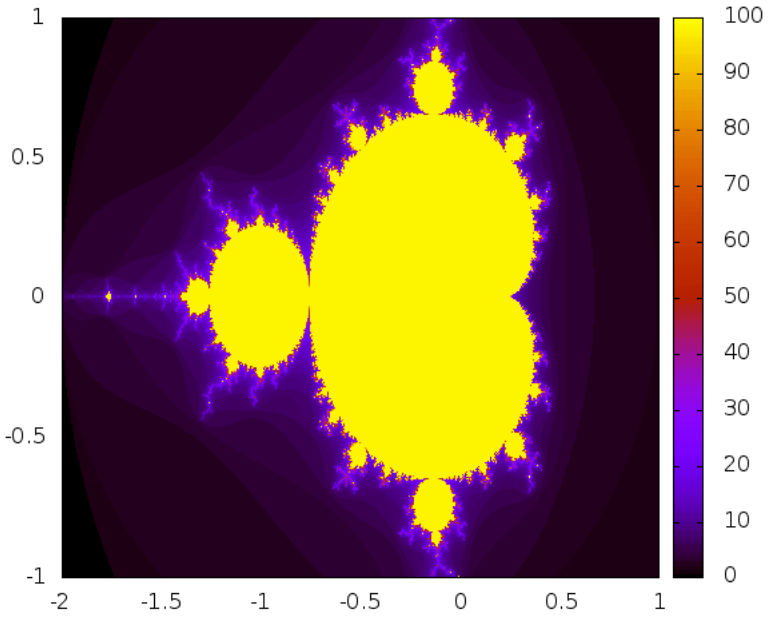

Figure 1: The Mandelbrot set plotted on the complex plane. The horizontal axis is the real axis and the vertical axis is the imaginary axis.

Workshop 8 looked at three different versions of the Mandelbrot calculation which had been parallelised using MPI. One of these programs used the manager-worker pattern to distribute work between MPI processes.

In the program the number of iterations at each point in the complex plane is stored as a two dimensional array called `nIter`. An element of the array is accessed using `nIter[i][j]` where the index `i` refers to the real axis and the index `j` refers to the imaginary axis. In the manager-worker version of the program the manger process tells the worker processes which value of the `i` index to work on. The worker processes loop over `j` to calculate a column of values in the complex plane (this is done in the `calc_vals` function). When they have completed one column they send a message to the manager process to request another `i` value to work on. Workers keep requesting work until all the values of `i` have been handed out. Each worker process stores the results it has calculated, and the results are collated at the end of the calculation using a call to `MPI_Reduce` (this is done in the `do_communication` function).

Using `MPI_Reduce` to collate the results sends more data than is required because each process which participates in the call sends the whole computational domain. The values in the domain will includes zeros where a process has not calculated any values. The amount of data sent can be minimised if each worker process sends its results back to the the manger process each time it

requesta a new `i` value. This means that each process only sends values it has calculated and there is no need to call `MPI_Reduce` to collate the results at the end of the calculation.

In this assignment you will modify the communication pattern in the manger-worker version of the Mandelbrot program so that worker processes send their results to the manger process after each column has been calculated instead of calling `MPI_Reduce` at the end of the calculation. You will then measure the impact of this optimisation on the parallel scaling of the program.

2 Optimising the communication pattern

Task 1: Modify the communication pattern in the manger-worker version of the Mandelbrot program so that worker processes send their results to the manger process after each column has been calculated instead of calling `MPI_Reduce` at the end of the calculation.

You have been provided with the original version of the manager-worker Mandelbrot program and for task 1 you need to optimise the communication pattern by making sure that the worker processes only send data that they have calculated. The following steps can be used to implement this optimisation:

1. Create buffer space for sending and receiving data. The buffer space needs to be the size of one column in the complex plane plus two extra values (see next point).

2. Modify the point-to-point communication calls so that the worker processes send their results back to the manger process after calculating each column (each `i` value). This can be done as part of the request for work: the worker process sends its rank, the `i` value of the completed column and the data in the column. These are all `int` values which can be packed into a single buffer.

3. You will need to handle the case of the initial handout where there is no data to be sent back to the manager process. This can be done by setting the `i` value of the column to a missing data value (e.g. a negative value) so that the manager process knows to discard the data.

4. The manager process receives the message, unpacks the column of values and stores it in the correct column of the `nIter` array. The manager process then hands out the next `i` value to the requesting worker process as before. When the calculation is complete the manager process will hold all the values in the `nIter` array.

5. You should now remove or comment out the call to `do_communication` as this is no longer required.

6. Write out the results from the manager process instead of the rank 1 process.

When you have modified the program you should check that the results match the results from the original program exactly. You should also check that all MPI processes exit cleanly after making a call to `MPI_Finalize`.

You may find the following advice useful:

- You do not have to use Isca to develop the MPI version of the program. This part of the assignment can be carried out using another computer with a C compiler and MPI library (for example the Lovelace lab PCs) if you prefer.

- The `diff --brief` command can be used to check whether the output from the modified version matches the output from the original version.

- If the output is not as expected it may help to initialise the `nIter` array to a missing data value (e.g. negative integer). This will make it clear where the array is not being filled with values.

- You may wish to decrease the problem size whilst testing the program but remember to change back to the original size for the performance tests in the next section.

3 Measuring the performance impact

Task 2: Measure the impact of the optimisation on the strong scaling of the calculation when run on Isca.

For this scaling test you should switch off I/O (by setting `doIO=false`) and use `N_RE=12000` and `N_IM=8000`. This part of the assignment needs to be carried out using the Isca HPC system. The workshop 8 tar file include job scripts which you can use to run these tests on Isca.

To carry out the scaling test follow these steps:

1. Run the original program using 2, 4, 8, 12 and 16 MPI process on one node, and 32 MPI processes on 2 nodes.
2. Run the modified program using 2, 4, 8, 12 and 16 MPI process on one node, and 32 MPI processes on 2 nodes.
3. Using your results calculate parallel speed-up data for the original and modified versions of the program. You may assume that the serial run time is 90.985 seconds (i.e. the value used in workshop 8).
4. Plot the parallel speed-up against the number of MPI processes for both the original and modified versions of the program.

4 Deliverables

There are two deliverables for this assignment:

1. The source code for your modified program. The submitted program should implement the point-to-point communication pattern described in these instructions and should produce output which identical to the original program. When the program is run all MPI processes should exit cleanly after making a call to `MPI_Finalize`.
2. A short report (guideline length 2–3 pages) which describes your modifications to the program and presents the results of your scaling study. The report should be presented in a consistent font no smaller than 11 point, with margins not less than 2cm, single column, and single space between lines. The report should include the following elements:
   (a) A title
   (b) A short introduction describing what the report is about
   (c) A description of your modifications to the program. You may include snippets of source code if required.
   (d) A plot showing the results from your scaling test. The plot should show parallel-speed up against number of MPI processes for both the original program and your modified version. The axes should be labelled and the two lines should be clearly distinguishable.
   (e) A paragraph describing and assessing your results. You should say whether your modifications have improved the parallel performance of the program and quantify the size of any improvement.

The deliverables should uploaded to BART a single zip or tar file containing the source code for the modified version of the program, and the report. The deadline for submission is 12 noon 29 March 2023.

5 Mark scheme

A total of 100 marks are available for this assignment:

1. **Task 1: Optimising the communication pattern (50 marks)**

   This task will be assessed based on the modified version of the program. Marks will be awarded according to the following criteria:

   - The program implements the required communication pattern correctly and the results are identical to the original program. When the program is run all MPI processes exit cleanly after making a call to `MPI_Finalize`: **50 marks**
   - The program implements the required communication pattern but a minor error causes the results to differ slightly from the original program. When the program is run all MPI processes exit cleanly after making a call to `MPI_Finalize`: **30 marks**
   - The program implements the required communication pattern but there is one major error. Either the results differ significantly from the original program **or** when the program is run all MPI processes do not exit cleanly after making a call to `MPI_Finalize`: **20 marks**
   - The program implements the required communication pattern but there are two major errors. The results differ significantly from the original program **and** when the program is run all MPI processes do not exit cleanly after making a call to `MPI_Finalize`: **10 marks**

2. **Task 2: Measuring the performance impact (50 marks)**

   This task will be assessed based on the on the report according to the following criteria:

   (a) The report includes a title and a short introduction describing what the report is about: **10 marks**

   (b) The report correctly describes all the modifications to the program: **10 marks**

   (c) The report includes a plot showing the results from the scaling test presenting data for both the original program and the modified version. The axes are labelled and the two lines are clearly distinguishable: **20 marks**

   (d) The report include a paragraph describing and assessing the results including a quantitive statement about whether the modifications have improved the parallel performance of the program: **10 marks**